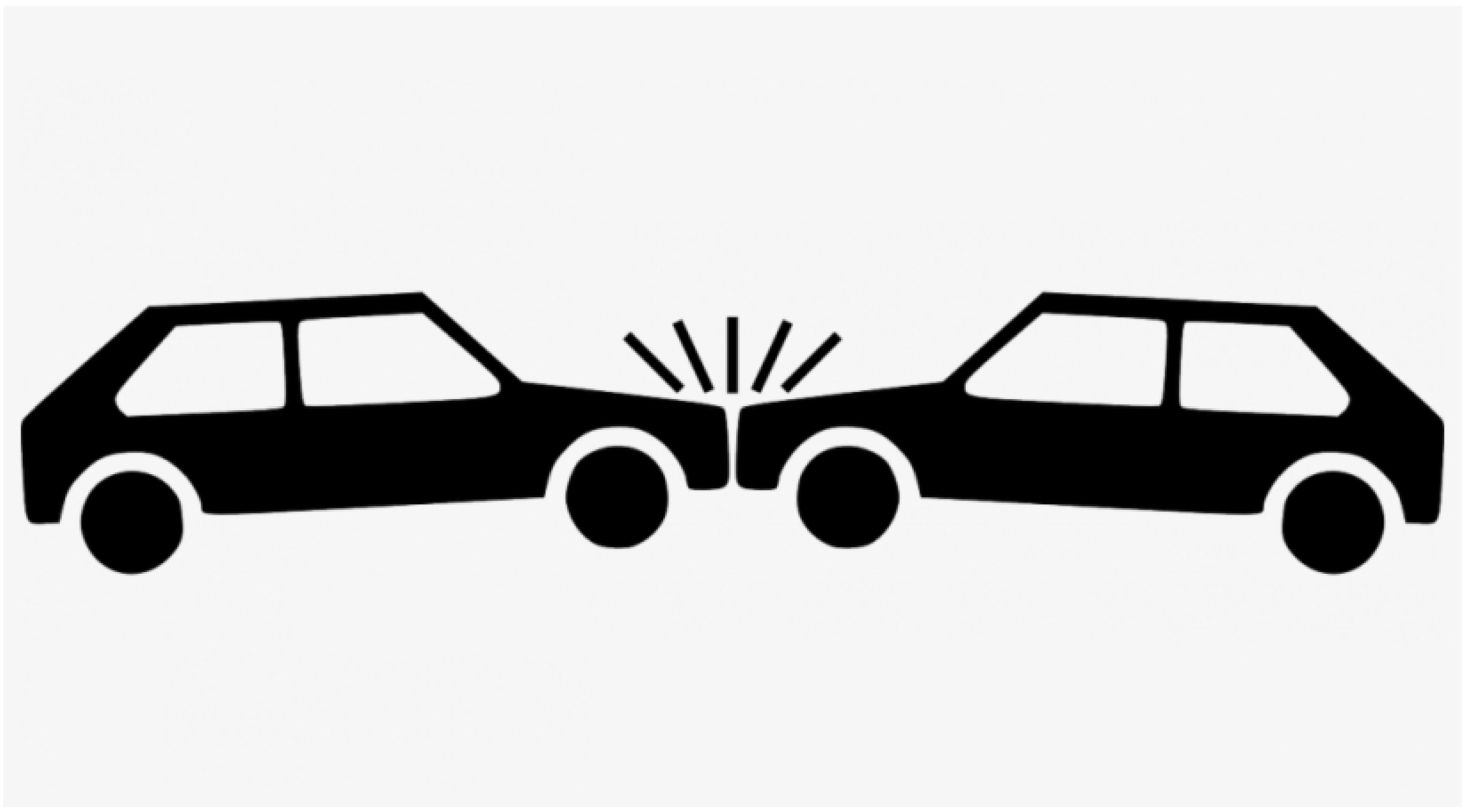
(2021-22 BNM832) Big Data for Decision Making Individual Assignment



Accident Severity Prediction Using Classification

In the group assignment, the UK Road Safety Dataset underwent the following :

1. The data was preprocessed
2. A new feature dataframe was created
3. The new dataframe was split for training & testing
4. Data imputation was done
5. Exploratory analysis of the features
6. One hot encoding

Table of Contents

1.0 Importing Essential Libraries and Preparing Environment

In [1]:
```
# Base Libraries
import re
import time
import numpy as np
import pandas as pd
import statistics
import warnings
warnings.filterwarnings(action='ignore')

# Library for Plotting
import seaborn as sns
sns.set(style="darkgrid")
import matplotlib.pyplot as plt
%matplotlib inline
```

2.0 Business Objective

Most car accident in the UK occur on high speed motorways in the countryside that are far from any hospital or emergency services reach. As a consequence, many chronic injuries are sustained, along with loss of life. Most injuries incured in car accidents can be prevented if proper care relief is provided at the right time. In times of multiple car accident crashes in the vicinity of the nearest emergency service, the aid has only a limited number of resources, which cannot practically be dispereseded to each accident site. A situation like this requires priority based response.

The objective of this project is to predict accident severity accurately based on predictor variables that can be obtained from the site of crash. The severity rating can then be used to prioritise dispatch of response team and organize first aid resources accordingly.

3.0 Data Loading

At the end of the group assignment, four dataframes were created, one for the train datasets and another one for the test dataset. Each set was split into two dataframes, one for the dependent variable y and the other one for the independent variable x.

In [2]:

```
# Loading .csv files into dataframes
x_train = pd.read_csv("x_accident_train.csv", index_col=0)
y_training = pd.read_csv("y_accident_train.csv", index_col=0)
x_test = pd.read_csv("x_accident_test.csv", index_col=0)
y_testing = pd.read_csv("y_accident_test.csv", index_col=0)
```

The train dataset's shape and types of their variables are analysed:

In [3]:

```
y_train = y_training["accident_severity"]
y_test = y_testing["accident_severity"]
```

In [4]:

```
x_train.head()
```

Out[4]:

	speed_limit	age_of_vehicle	engine_capacity_cc	latitude	longitude	first_point_of_impact_1	first_point_of_impact_2	first_point_of_impact_3	first_point_
0	60.0	4.0	1499.0	52.495285	0.580066	0.0	0.0	0.0	
1	30.0	16.0	1598.0	51.168324	-0.176178	0.0	0.0	0.0	
2	40.0	6.0	1398.0	52.491563	-0.722810	0.0	1.0	0.0	
3	40.0	8.0	1560.0	51.513778	-2.694186	1.0	0.0	0.0	
4	30.0	8.0	1560.0	51.382694	1.375307	1.0	0.0	0.0	

5 rows × 101 columns

In [5]:

```
x_test.head()
```

Out[5]:

	speed_limit	age_of_vehicle	engine_capacity_cc	latitude	longitude	first_point_of_impact_1	first_point_of_impact_2	first_point_of_impact_3	first_point_
0	30.0	9.0	2499.0	51.406930	-1.312713	0.0	0.0	0.0	
1	40.0	6.0	998.0	51.678997	-3.876258	1.0	0.0	0.0	
2	30.0	8.0	1560.0	53.761235	-1.774380	1.0	0.0	0.0	
3	40.0	10.0	1596.0	52.393291	-1.472695	1.0	0.0	0.0	
4	30.0	13.0	1364.0	51.437921	0.173505	0.0	1.0	0.0	

5 rows × 101 columns

The dataframe is in this manner due to the usage of One hot encoding during the group assignment

In [6]:

```
y_train.head()
```

Out[6]:

```
49685    3
66084    3
45453    2
82737    3
72406    3
Name: accident_severity, dtype: int64
```

In [7]:

```
y_test.head()
```

Out[7]:

```
58678    3
88128    3
20691    3
32670    2
69746    3
Name: accident_severity, dtype: int64
```

In [8]:

```
print(x_train.shape)
print(y_train.shape)
```

```
(64674, 101)
(64674,)
```

4.0 Variable Correlations

An analysis is done to find out which variables have a correlation with the target variable "accident_severity". Pearson's r-correlation technique will be used and the correlation coefficients will be sorted from strongest to weakest.

In [9]:
```
corr_matrix = x_train.corrwith(y_train, axis=0,  method='pearson')
corr_matrix[y_train].sort_values(ascending=False)
```

Out[9]:
```
age_of_vehicle    0.002977
age_of_vehicle    0.002977
age_of_vehicle    0.002977
age_of_vehicle    0.002977
age_of_vehicle    0.002977
                    ...   
latitude         -0.001069
latitude         -0.001069
latitude         -0.001069
latitude         -0.001069
latitude         -0.001069
Length: 64674, dtype: float64
```

When the correlation list are analysed, we notice that a lot of the features have a very low correlation score with accident_severity

5.0 Feature selection

After analyzing the correlations between the features, the following features are dropped due to low correlation scores.

In [10]:
```
# Dropping features from training data

x_train = x_train.drop(['latitude', 'longitude','vehicle_location_restricted_lane_6',

                       'age_band_of_casualty_6', 'vehicle_location_restricted_lane_3',

                       'speed_limit', 'driver_imd_decile_6', 'vehicle_leaving_carriageway_8',

                       'driver_imd_decile_5', 'weather_conditions_3', 'age_band_of_casualty_3',

                       'junction_location_1', 'first_point_of_impact_4', 'driver_imd_decile_9',

                       'driver_imd_decile_3', 'junction_location_9', 'trunk_road_flag_2',

                       'age_of_vehicle', 'light_conditions_7', 'driver_imd_decile_4',

                       'vehicle_leaving_carriageway_3', 'skidding_and_overturning_2',

                       'weather_conditions_2'], axis=1)

# Dropping features from testing data

x_test = x_test.drop(['latitude', 'longitude','vehicle_location_restricted_lane_6',

                       'age_band_of_casualty_6', 'vehicle_location_restricted_lane_3',

                       'speed_limit', 'driver_imd_decile_6', 'vehicle_leaving_carriageway_8',

                       'driver_imd_decile_5', 'weather_conditions_3', 'age_band_of_casualty_3',

                       'junction_location_1', 'first_point_of_impact_4', 'driver_imd_decile_9',

                       'driver_imd_decile_3', 'junction_location_9', 'trunk_road_flag_2',

                       'age_of_vehicle', 'light_conditions_7', 'driver_imd_decile_4',

                       'vehicle_leaving_carriageway_3', 'skidding_and_overturning_2',

                       'weather_conditions_2'], axis=1)
```

In [11]:
```
# Shape of datesets after feature selection
print(x_train.shape)
print(x_test.shape)
```

```
(64674, 78)
(27718, 78)
```

6.0 Scaling

The dataset is standardized to improve the performance of the models. The original values of the target variable is kept.

The training dataset is then scaled using StandardScaler which standardizes a feature by subtracting the mean and then scaling to unit variance.

In [12]:
```
# importing standard scaler libraries
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()

# fit_transform returns Numpy array, so it needs to be put it back into dataframe
scaled = scaler.fit_transform(x_train)
scaled_train = pd.DataFrame(scaled, columns=x_train.columns)
```

In [13]:
```
# Scaled training dataset
scaled_train.head()
```

Out[13]:

	engine_capacity_cc	first_point_of_impact_1	first_point_of_impact_2	first_point_of_impact_3	urban_or_rural_area_2	age_band_of_casualty_2	age_band_of_ca
0	-0.120908	-1.052266	-0.449204	-0.39983	1.173652	-0.155817	-
1	-0.023230	-1.052266	-0.449204	-0.39983	-0.852042	-0.155817	-
2	-0.220559	-1.052266	2.226162	-0.39983	-0.852042	-0.155817	-
3	-0.060722	0.950330	-0.449204	-0.39983	1.173652	-0.155817	-
4	-0.060722	0.950330	-0.449204	-0.39983	-0.852042	-0.155817	-

5 rows × 78 columns

Similarly, the testing dataset is also scaled using StandScaler.

In [14]:
```
# importing standard scaler libraries
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# fit_transform returns a Numpy array, so we need to put it back into dataframe

scaled = scaler.fit_transform(x_test)
scaled_test = pd.DataFrame(scaled, columns=x_test.columns)
```

In [15]:
```
# Scaled testing dataset
scaled_test.head()
```

Out[15]:

	engine_capacity_cc	first_point_of_impact_1	first_point_of_impact_2	first_point_of_impact_3	urban_or_rural_area_2	age_band_of_casualty_2	age_band_of_ca
0	0.836305	-1.054976	-0.447117	-0.394877	-0.851054	-0.158706	-
1	-0.603641	0.947889	-0.447117	-0.394877	1.175014	-0.158706	-
2	-0.064501	0.947889	-0.447117	-0.394877	-0.851054	-0.158706	-
3	-0.029965	0.947889	-0.447117	-0.394877	1.175014	-0.158706	-
4	-0.252529	-1.054976	2.236552	-0.394877	-0.851054	-0.158706	-

5 rows × 78 columns

7.0 Model Building

For the purpose of building a machine learnig model, five algorithms are chosen and they will be trained on the training dateset and their results will be compared before picking the best ones to evaluate on the test dataset.

Here are the chosen machine learning algorithms for our model:

1. Naïve Bayes Algorithm
2. XGBoost Classifier
3. Support Vector Machine
4. K Nearest Neighbour
5. Logistic Regression

7.1 Baseline

The baseline for the project is calculated using micro-averaging where precision, recall and F-score are evaluated in each class seperately and then averaged across classes.

For the "Slight (3)" label, the accuracy measures will be :

Precision: ((True Positive)/((True Positive) + (False Posititve))) : 48868/64674 = 0.7556

Recall: ((True Positive)/((True Positive) + (False Negative))) : 48868/48868 = 1.0

F-score: 2/(1/precision + 1/recall) = 0.86

Therefore, the baseline is 0.86.

8.0 Training and Evaluating the Models

The models are evaluated using the training data and the model with the best results are used to evaluate the test data.

8.1 Naïve Bayes Algorithm

In [16]:
```
# Importing Library
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

model1 = GaussianNB()
model1.fit(x_train, y_train)
print("Naïve Bayes Model fitting completed.\n")

print("Training Dataset Prediction:\n")

print('*_'*20+ "Naïve Bayes Algorithm" +'_*'*20)
model = model1
y_pred = model.predict(x_train)
arg_test = {'y_true':y_train, 'y_pred':y_pred}
print(confusion_matrix(**arg_test))
print(classification_report(**arg_test))
```

```
Naïve Bayes Model fitting completed.

Training Dataset Prediction:

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_Naïve Bayes Algorithm_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
[[  811   397   168]
 [ 2428  9549  2453]
 [ 4574   326 43968]]
              precision    recall  f1-score   support

           1       0.10      0.59      0.18      1376
           2       0.93      0.66      0.77     14430
           3       0.94      0.90      0.92     48868

    accuracy                           0.84     64674
   macro avg       0.66      0.72      0.62     64674
weighted avg       0.92      0.84      0.87     64674
```

8.2 XGBoost Classifier

In [17]:
```
!pip install xgboost
import xgboost

model2 = xgboost.XGBClassifier(verbosity = 0)
model2.fit(x_train, y_train)
print("XGBoost Classifier Model fitting completed.\n")

print("Training Dataset Prediction:\n")

print('*_'*20+ "XGBoost Classifier" +'_*'*20)
model = model2
y_pred = model.predict(x_train)
arg_test = {'y_true':y_train, 'y_pred':y_pred}
print(confusion_matrix(**arg_test))
print(classification_report(**arg_test))
```

```
Requirement already satisfied: xgboost in c:\programdata\anaconda3\lib\site-packages (1.5.2)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from xgboost) (1.20.3)
XGBoost Classifier Model fitting completed.

Training Dataset Prediction:

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_XGBoost Classifier_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
[[ 1016   168   192]
 [    0 11702  2728]
 [    0     2 48866]]
              precision    recall  f1-score   support

           1       1.00      0.74      0.85      1376
           2       0.99      0.81      0.89     14430
           3       0.94      1.00      0.97     48868

    accuracy                           0.95     64674
   macro avg       0.98      0.85      0.90     64674
weighted avg       0.95      0.95      0.95     64674
```

8.3 Support Vector Machine

In [18]:
```
 # Importing Library
from sklearn.svm import SVC

model3 = SVC(kernel = 'sigmoid', gamma='scale')
```

```
model3.fit(x_train, y_train)
print("Support Vector Machine Model fitting completed.\n")

print("raining Dataset Prediction:\n")

print('*_'*20+ "Support Vector Machine Model" +'_*'*20)
model = model3
y_pred = model.predict(x_train)
arg_test = {'y_true':y_train, 'y_pred':y_pred}
print(confusion_matrix(**arg_test))
print(classification_report(**arg_test))
```

```
Support Vector Machine Model fitting completed.

raining Dataset Prediction:

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_Support Vector Machine Model_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
[[   97   240  1039]
 [  365  2785 11280]
 [ 1031 10518 37319]]
              precision    recall  f1-score   support

           1       0.06      0.07      0.07      1376
           2       0.21      0.19      0.20     14430
           3       0.75      0.76      0.76     48868

    accuracy                           0.62     64674
   macro avg       0.34      0.34      0.34     64674
weighted avg       0.62      0.62      0.62     64674
```

8.4 K Nearest Neighbour

In [19]:
```
# Importing Library
from sklearn.neighbors import KNeighborsClassifier

model4 = KNeighborsClassifier()
model4.fit(x_train, y_train)
print("K Nearest Neighbour Model fitting completed.\n")

print("Training Dataset Prediction:\n")

print('*_'*20+ "K-Nearest Neighbour" +'_*'*20)
model = model4
y_pred = model.predict(x_train)
arg_test = {'y_true':y_train, 'y_pred':y_pred}
print(confusion_matrix(**arg_test))
print(classification_report(**arg_test))
```

```
K Nearest Neighbour Model fitting completed.

Training Dataset Prediction:

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_K-Nearest Neighbour_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
[[  443   323   610]
 [   58 11224  3148]
 [   30   308 48530]]
              precision    recall  f1-score   support

           1       0.83      0.32      0.46      1376
           2       0.95      0.78      0.85     14430
           3       0.93      0.99      0.96     48868

    accuracy                           0.93     64674
   macro avg       0.90      0.70      0.76     64674
weighted avg       0.93      0.93      0.93     64674
```

8.5 Logistic Regression

In [20]:
```
# Importing Library
from sklearn.linear_model import LogisticRegression

model5 = LogisticRegression()
model5.fit(x_train, y_train)
print("Logistic Regression Model fitting completed.\n")

print("Training Dataset Prediction:\n")

print('*_'*20+ "Logistic Regression" +'_*'*20)
model = model5
y_pred = model.predict(x_train)
arg_test = {'y_true':y_train, 'y_pred':y_pred}
print(confusion_matrix(**arg_test))
print(classification_report(**arg_test))
```

```
Logistic Regression Model fitting completed.

Training Dataset Prediction:

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_Logistic Regression_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
```

```
[[   32  1080   264]
 [    0 11480  2950]
 [    3     0 48865]]
              precision    recall  f1-score   support

           1       0.91      0.02      0.05      1376
           2       0.91      0.80      0.85     14430
           3       0.94      1.00      0.97     48868

    accuracy                           0.93     64674
   macro avg       0.92      0.61      0.62     64674
weighted avg       0.93      0.93      0.92     64674
```

From the five model which were evaluated, the XGBoost Classifier had the best results and is chosen as the model to evaulate the test dataset.

9.0 XGBoost Classifer

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. XGBoost has become the leading and the most preferred machine learning algorithm among data scientists in the 21st century, as it has been battle tested for production on large-scale problems.

It is a highly flexible and versatile tool that can work through most regression, classification and ranking problems as well as user-built objective functions and is perfect of the dataset we are currently working on.

9.1 Hyperparameter Tuning

The model has displayed a high level of accuracy but it can be improved further by using Hyperparameter Tuning.

The hyperparameters are tuned using the random search method. For that, we'll use the sklearn library, which provides a function specifically for this purpose: RandomizedSearchCV.

RandomizedSearchCV randomly passes the set of hyperparameters and calculate the score and gives the best set of hyperparameters which gives the best score as an output. This is used to tune the model further.

In [21]:
```
#Hyperparameter optimization using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
classifier = xgboost.XGBClassifier(verbosity = 0)
```

So, initially we create a dictionary of some parameters to be trained upon. Here the keys are basically the parameters and the values are a list of values of the parameters to be trained upon. So the RandomizedSearchCV will test each value and find out the particular value which gives the highest accuracy.

In [22]:
```
params = {'max_depth': [3, 6, 10, 15],
              'learning_rate': [0.01, 0.1, 0.2, 0.3, 0.4],
              'subsample': np.arange(0.5, 1.0, 0.1),
              'colsample_bytree': np.arange(0.5, 1.0, 0.1),
              'colsample_bylevel': np.arange(0.5, 1.0, 0.1),
              'n_estimators': [100, 250, 500, 750],
              'num_class': [10]
              }
```

RandomizedSearchCV() is called and the parameter above were passed below:

In [23]:
```
xgbclf = xgb.XGBClassifier(objective="multi:softmax", tree_method='hist')
classifier = RandomizedSearchCV(estimator=xgbclf,
                         param_distributions=params,
                         scoring='accuracy',
                         n_iter=5,
                         n_jobs=-1,
                         verbose=0)
```

The model is then fitted with the training data:

In [24]:
```
classifier.fit(x_train,y_train)
```

Out[24]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           enable_categorical=False, gamma=None,
                                           gpu_id=None, importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None...
                                           validate_parameters=None,
                                           verbosity=None),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'colsample_bylevel': array([0.5, 0.6, 0.7, 0.8, 0.9]),
                                        'colsample_bytree': array([0.5, 0.6, 0.7, 0.8, 0.9]),
                                        'learning_rate': [0.01, 0.1, 0.2, 0.3,
                                                          0.4],
                                        'max_depth': [3, 6, 10, 15],
                                        'n_estimators': [100, 250, 500, 750],
                                        'num_class': [10],
                                        'subsample': array([0.5, 0.6, 0.7, 0.8, 0.9])},
                   scoring='accuracy')
```

After the model has been fitted, it lets us look into all the parameters that have been selected by the RandomizedSearch() for theXGBClassifier. We can do it with the help of the best_estimators_method.

In [25]:
```
# parameters selected
classifier.best_estimator_
```

Out[25]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.5,
              colsample_bynode=1, colsample_bytree=0.8999999999999999,
              enable_categorical=False, gamma=0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.01, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=750, n_jobs=8, num_class=10, num_parallel_tree=1,
              objective='multi:softprob', predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=0.7,
              tree_method='hist', validate_parameters=1, verbosity=None)
```

After knowing all the best parameters, we can simply build our final classifier model by passing all those parameters.

In [26]:
```
# Building final classifier model
classifier= xgboost.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.7,
              colsample_bynode=1, colsample_bytree=0.7999999999999999,
              enable_categorical=False, gamma=0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.2, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, monotone_constraints='()',
              n_estimators=250, n_jobs=8, num_class=10, num_parallel_tree=1,
              objective='multi:softprob', predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=None,
              subsample=0.8999999999999999, tree_method='hist',
              validate_parameters=1, verbosity=0)
```

10. Evaluating the Test Dataset

The XGBoost Classifier Model which has been tuned using RandomizedSearchCV is used to evaluvate the test dataset.

In [27]:
```
print("Test Set Prediction for XGBoost Classifier:\n")

print('*_'*20+ "XGBoost Classifier" +'_*'*20)
model = classifier
y_pred = model.predict(x_test)
arg_test = {'y_true':y_test, 'y_pred':y_pred}
print(confusion_matrix(**arg_test))
print(classification_report(**arg_test))
```

```
Test Set Prediction for XGBoost Classifier:

*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_XGBoost Classifier_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
[[  388    90    98]
 [    0  5080  1209]
 [    2     9 20842]]
              precision    recall  f1-score   support

           1       0.99      0.67      0.80       576
           2       0.98      0.81      0.89      6289
           3       0.94      1.00      0.97     20853

    accuracy                           0.95     27718
   macro avg       0.97      0.83      0.89     27718
weighted avg       0.95      0.95      0.95     27718
```

It can be seen that the tuned model has given a high accuracy results as well after evaluating the testset data.

11. Cross-validation:

The model is then cross validated to test it's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset.

In [28]:
```
from sklearn.model_selection import cross_val_score

scores=cross_val_score(classifier,x_train,y_train,cv=10)
print("\n")
print('Cross-validation scores:{}'.format(scores))
print("")
print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Cross-validation scores:[0.94897959 0.94727891 0.95037106 0.94882498 0.95237359 0.94417814
 0.94587908 0.95376527 0.95051801 0.94912633]

Average cross-validation score: 0.9491
```

Using the mean cross-validation, we can conclude that we expect the model to be around 95% accurate on average.

The 10 scores produced by the 10-fold cross-validation shows that there is low variance in the accuracy between folds. So we can conclude that the model is independent of the particular folds used for training

K-Fold Cross-validation

To build more robust models with XGBoost, K-fold cross validation is always performed.

In this way, we ensure that the original training dataset is used for both training and validation.

We can compute the root mean squared error in order to evaluate the performance of our model:

In [29]:

```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE: %f" % (rmse))
```

```
RMSE: 0.248235
```

Based on a rule of thumb, it can be said that RMSE values between 0.2 and 0.5 shows that the model can relatively predict the data accurately.

In order to take full advantage of XGBoost's performance and efficiency, we convert the dataset into a DMatrix. This is achieved by using XGBoost's Dmatrix functionality.

In [30]:

```
data_dmatrix = xgb.DMatrix(data=x_train,label=y_train)
```

It's a very common practice to use cross validation to make models more robust. XGboost supports K-fold validation via the cv() functionality. We'll use this to apply cross validation to our model.

In [31]:

```
from xgboost import cv

params = {"objective":"reg:linear",'colsample_bytree': 0.3,'learning_rate': 0.1,
                'max_depth': 5, 'alpha': 10}

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=5,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse", as_pandas=True, seed=100)
```

The cv_results variable will return the train and test RMSE for each boosting round. The final boosting round metric can be obtained as follows:

In [32]:

```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
49    0.256915
Name: test-rmse-mean, dtype: float64
```

This shows that the model is accurate and very reliable.

12. Feature Importance with XGBoost

XGBoost provides a way to examine the importance of each feature in the original dataset within the model.

It involves counting the number of times each feature is split on across all boosting trees in the model.

Then we visualize the result as a bar graph, with the features ordered according to how many times they appear.

Then we can visualize the features that has been given the highest important score among all the features.

In [34]:

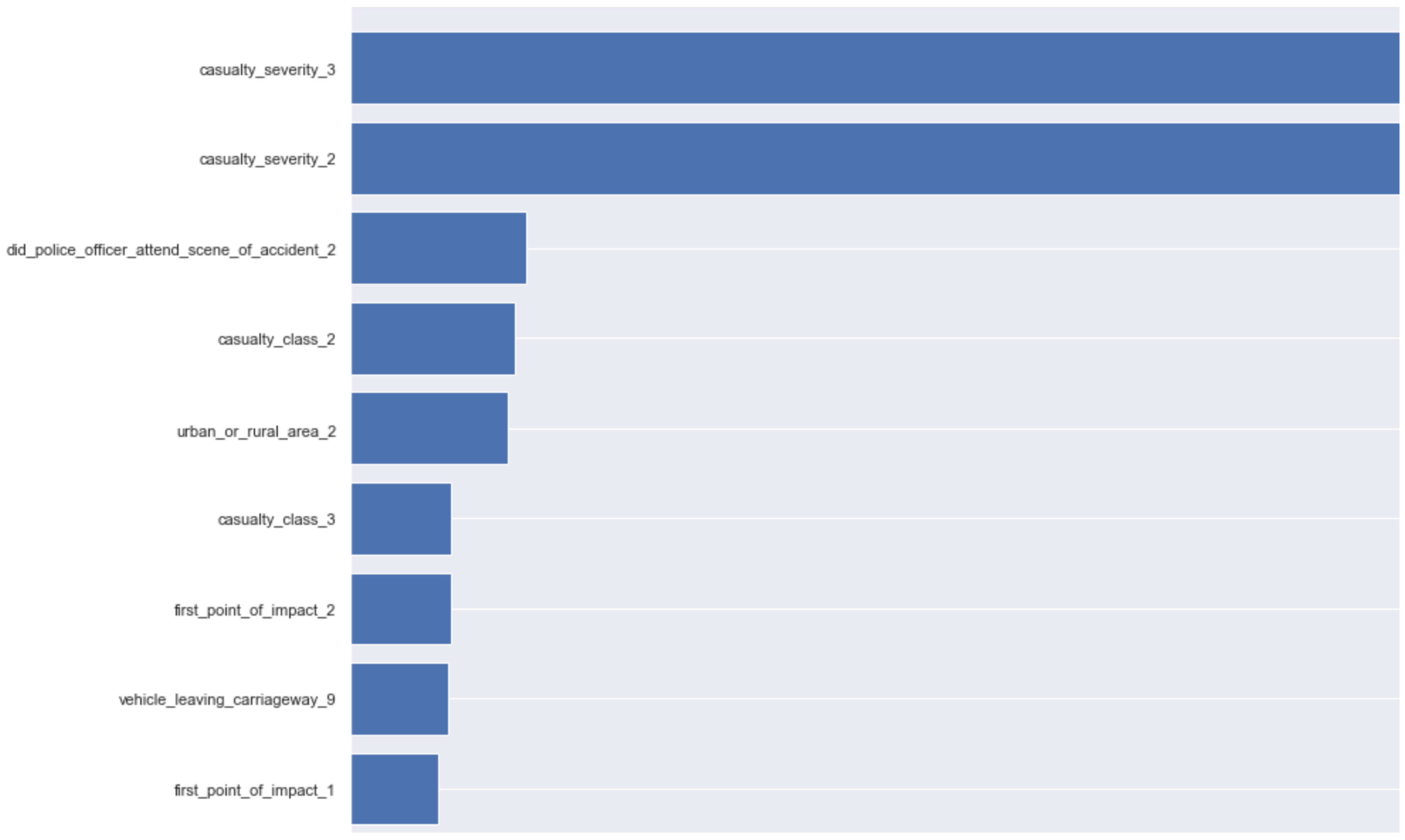
```
feature_importance = classifier.feature_importances_
sorted_idx = np.argsort(feature_importance)
fig = plt.figure(figsize=(200,100))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), np.array(x_test.columns)[sorted_idx])
plt.title('Feature Importance')
```

Out[34]:

```
Text(0.5, 1.0, 'Feature Importance')
```

From the graph it can be seen that the features with the most importance are:

1. casualty severity
2. did_police_officer_attend_scene_of_accident
3. casualty_class
4. urban_or_rural_area
5. first_point_of_impact
6. vehicle_leaving_carriageway etc.



A zoomed in version of the first part of the graph is shown above as the original graph is too huge due the large number of features.

The following features if thought about in real life scenarios are perfect ways to measure the severity of an accident.

For example, if casualty severity is high then naturally accident severity will be high as well, if a police officer had to show up to an accident site, then the severity has to be high, similarly urban areas would have more accidents and a feature like first point of impact can easily determine how severe an accident is.

13. Conclusion

This individual project was successful in implementing and evaluating several alternative models to address the business problem selected in the group assignment.

The baseline for the models was calculated and the training data was evaluated using five different classification algorithms.

Out of the various models that were evaluated the XGBoost Classifier had the best results when it was evaluated using training data with an accuracy of around 95 %.

The selected model was then made more accurate using hyperparameter tuning function called RandomSearchCV and the tuned model was used to evaluate the test dataset which also gave a results with 95 % accuracy.

The model was then cross-validated using the cross_val_score helper function and the K-Fold function available through XGBoost.

The feature importance order was analysed for finding possible real life applications and finally in the next section, possible future improvements and business scenarios are discussed.

14. Possible Future Improvements and Business Scenarios

Since there are accidents happening on a daily basis resulting in new data, we will have to re-merge the training and test sets regularly and retrain the model. This would defenitely improve the accuracy of the model and may even be able to find new patterns occuring within the data.

This model could be used by multiple other businesses, it can be used by insurance companies when they give out health insurance or car insurance plans for customers and they can charge higher prices depending on the risk the customer bring along with him or her which can be calculated their personal details with the model.

It can be used to rate the general safety of a new vehicle or calculate how safe an area is for traffic and pedestrians. The applications are endless.

With technology constantly evolving, new techniques and algorithms may be devised and we can improve our model further by adopting them in the future.

```
In [ ]:
```